



ITRI613 Databases I

Chapter 1



Learning outcomes

After engaging with the materials and activities in this study unit you should be able to:

- give an overview of the traditional file system as managed by the operating system, as opposed to the management by a DBMS;
- describe the levels of abstraction in a DBMS;
- know the architecture of a DBMS and name the different components.

What Is a DBMS?

- A program to manage a very large, integrated collection of data.
- Models real-world enterprise.
 - Entities (e.g., students, courses)
 - Relationships (e.g., Madonna is taking CS564)
- A Database Management System (DBMS) is a software package designed to store and manage databases.

Files vs. DBMS

- Application must stage large datasets between main memory and secondary storage (e.g., buffering, page-oriented access, 32-bit addressing, etc.)
- Special code for different queries
- Must protect data from inconsistency due to multiple concurrent users
- Crash recovery
- Security and access control

Why Use a DBMS?

- Data independence and efficient access.
- Reduced application development time.
- Data integrity and security.
- Uniform data administration.
- Concurrent access, recovery from crashes.

?

Why Study Databases??

- Shift from computation to information
 - at the “low end”: scramble to webspace (a mess!)
 - at the “high end”: scientific applications
- Datasets increasing in diversity and volume.
 - Digital libraries, interactive video, Human Genome project, EOS project
 - ... need for DBMS exploding
- DBMS encompasses most of CS
 - OS, languages, theory, AI, multimedia, logic

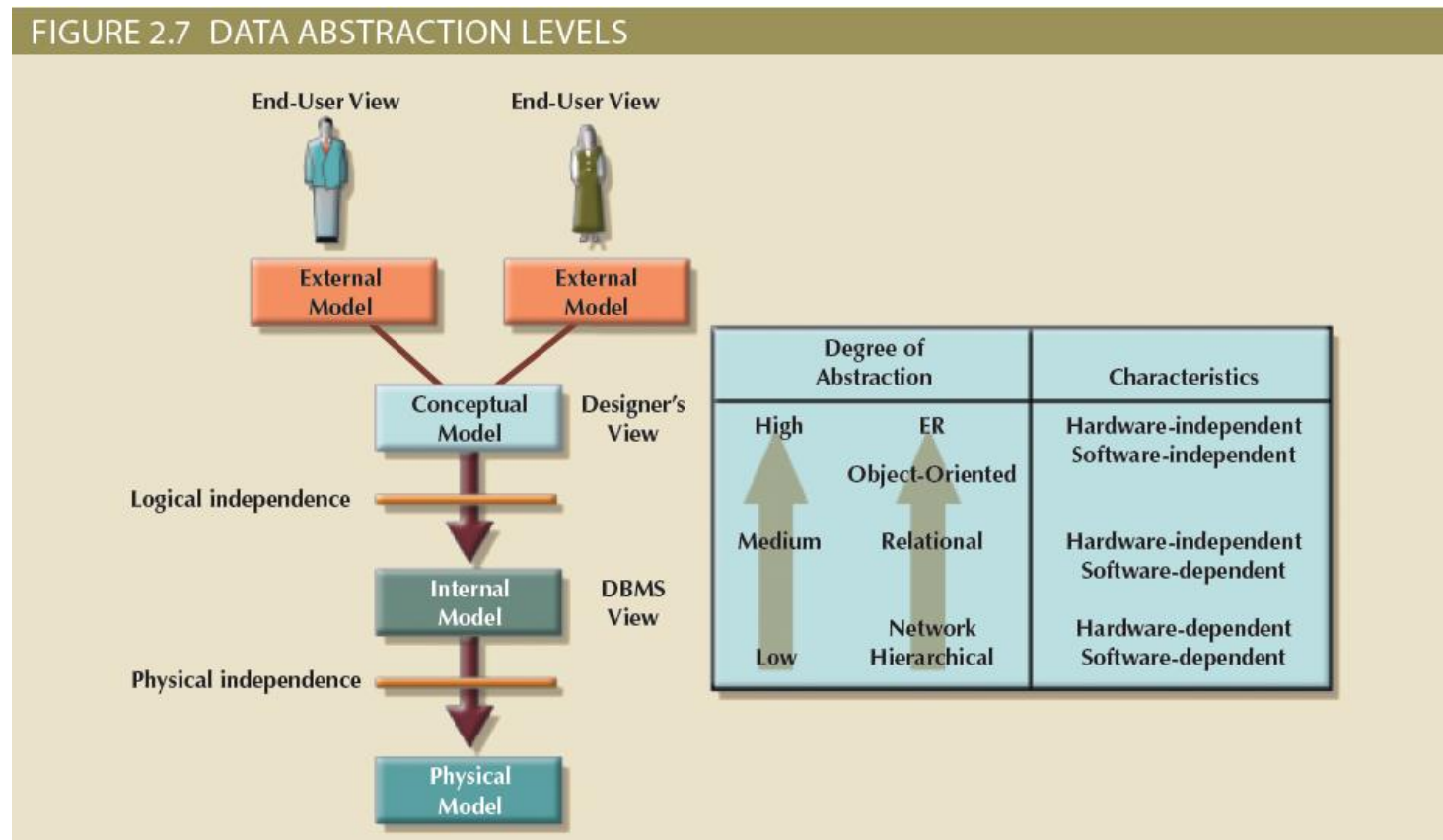
Data Models

- A *data model* is a collection of concepts for describing data.
- A *schema* is a description of a particular collection of data, using the a given data model.
- The *relational model of data* is the most widely used model today.
 - Main concept: *relation*, basically a table with rows and columns.
 - Every relation has a *schema*, which describes the columns, or fields.

Data Abstraction

- Data abstraction is the **reduction** of a particular body of data **to a simplified representation of the whole**
- Abstraction, in general, is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics

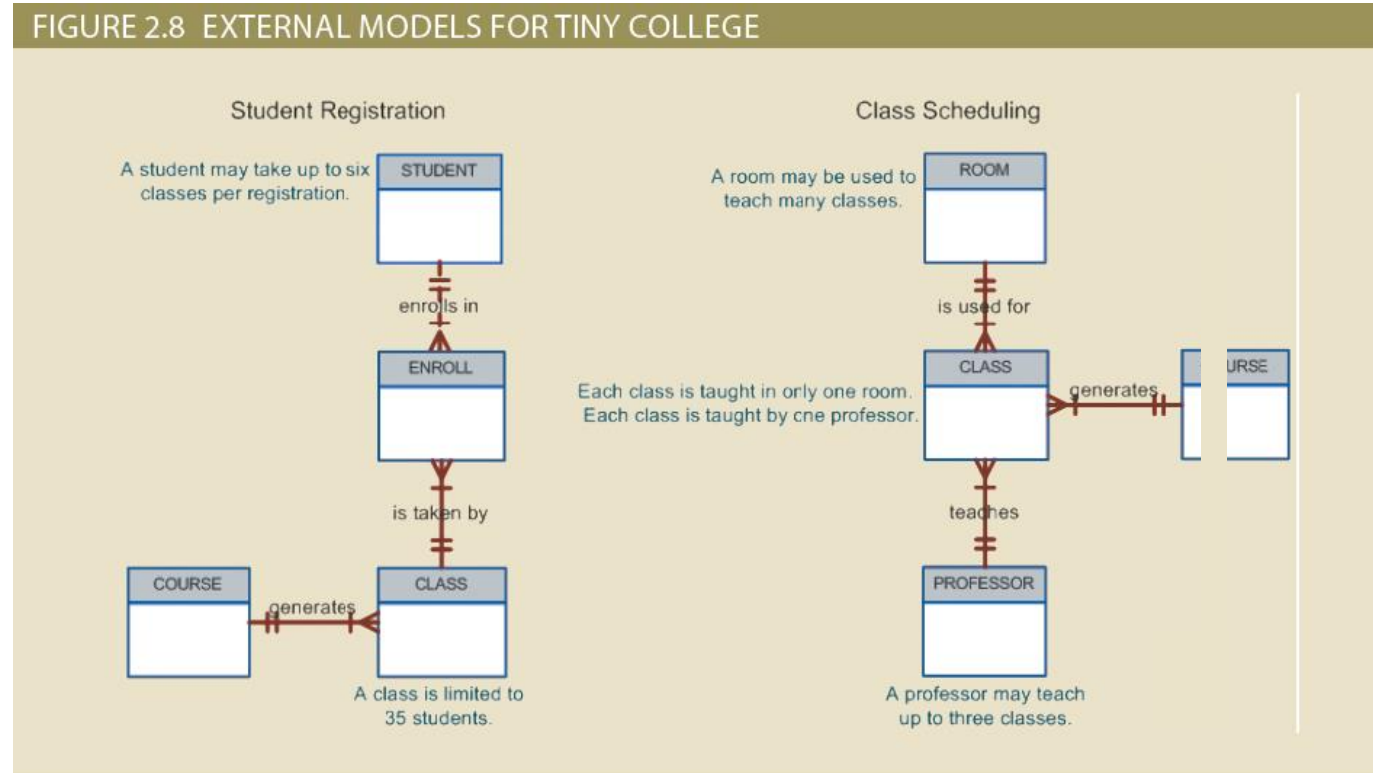
Figure 2.7 - Data Abstraction Levels



The External Model

- End users' view of the data environment
- ER diagrams are used to represent the external views
- **External schema:** Specific representation of an external view

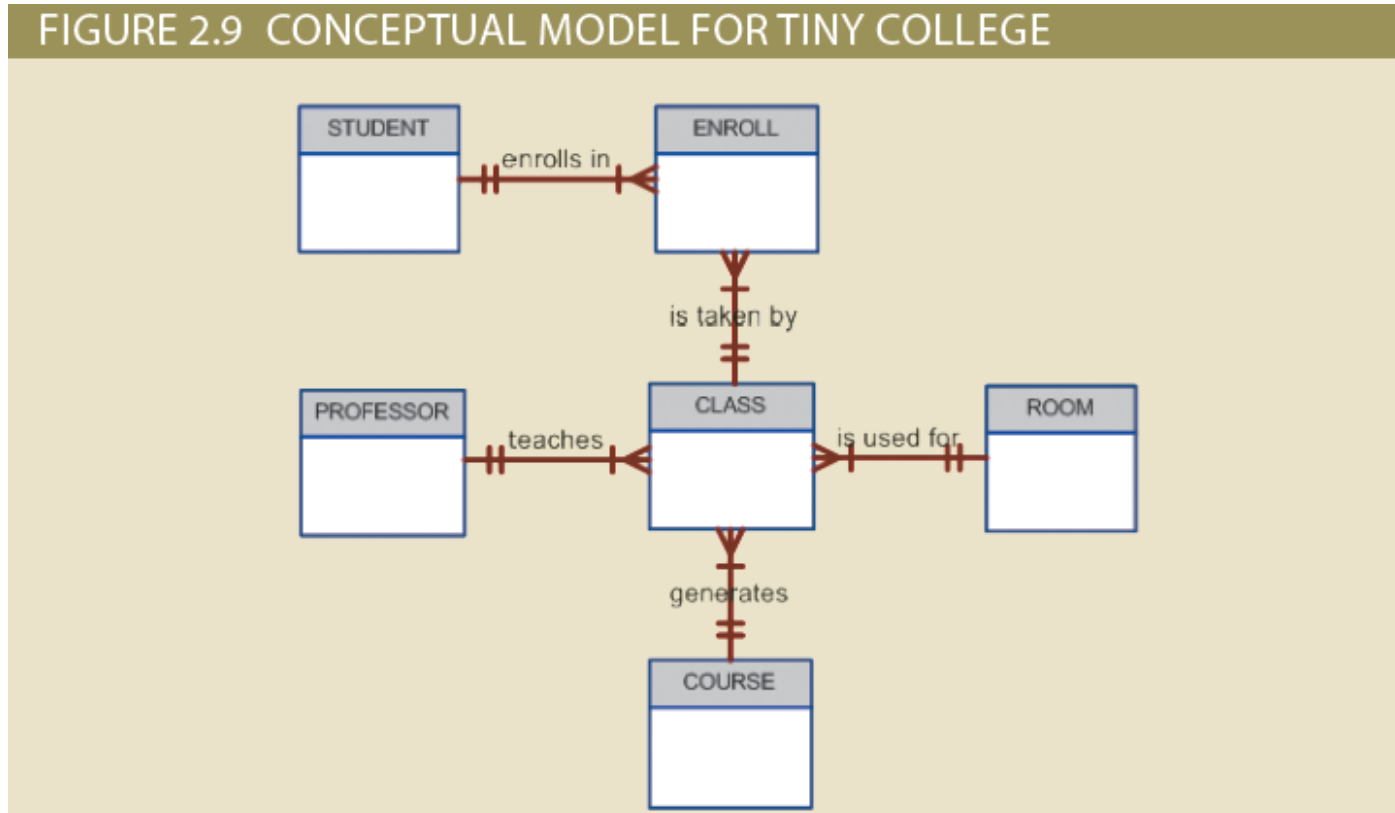
Figure 2.8 - External Models For Tiny College



The Conceptual Model

- Represents a global view of the entire database by the entire organization
- **Conceptual schema:** Basis for the identification and high-level description of the main data objects
- Has a macro-level view of data environment
- Is software and hardware independent
- **Logical design:** Task of creating a conceptual data model

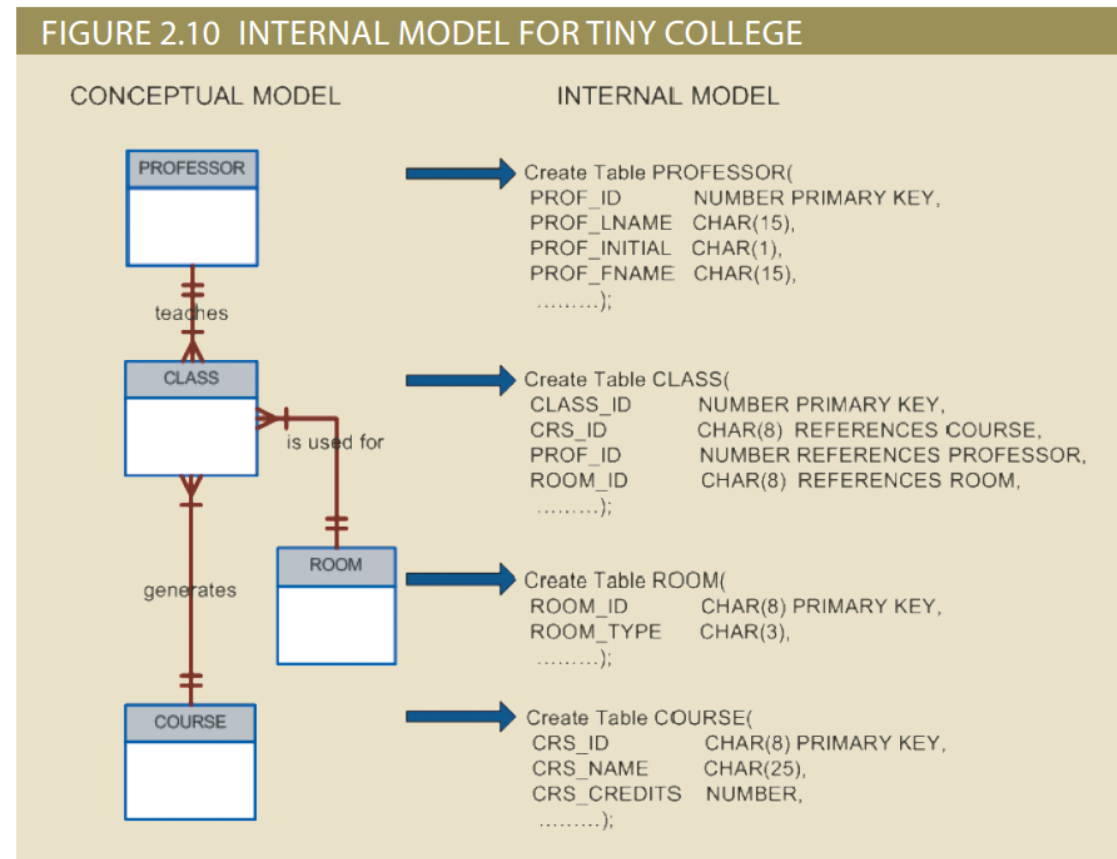
Figure 2.9 - Conceptual Model For Tiny College



The Internal Model

- Representing database as seen by the DBMS, mapping conceptual model to the DBMS
- **Internal schema:** Specific representation of an internal model
 - Uses the database constructs supported by the chosen database
- Is software dependent and hardware independent
- **Logical independence:** Changing internal model without affecting the conceptual model

Figure 2.10 - Internal Model for Tiny College




The Physical Model

- Operates at lowest level of abstraction
- Describes the way data are saved on storage media such as disks or tapes
- Requires the definition of physical storage and data access methods
- Relational model aimed at logical level
 - Does not require physical-level details
- **Physical independence:** Changes in physical model do not affect internal model

Table 2.4 - Levels of Data Abstraction

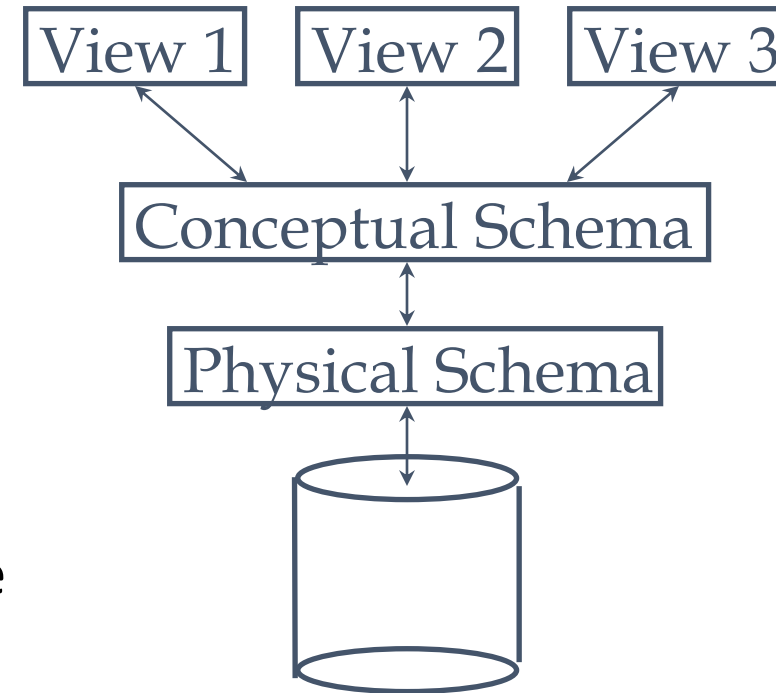
TABLE 2.4

LEVELS OF DATA ABSTRACTION

MODEL	DEGREE OF ABSTRACTION	FOCUS	INDEPENDENT OF
External	High  Low	End-user views	Hardware and software
Conceptual		Global view of data (database model independent)	Hardware and software
Internal		Specific database model	Hardware
Physical		Storage and access methods	Neither hardware nor software

Levels of Abstraction

- Many views, single conceptual (logical) schema and physical schema.
 - Views describe how users see the data.
 - Conceptual schema defines logical structure
 - Physical schema describes the files and indexes used.



** Schemas are defined using DDL; data is modified/queried using DML.*

Example: University Database

- Conceptual schema:
 - *Students(sid: integer, name: string, login: string, age: integer, gpa: real)*
 - *Courses(cid: integer, cname: string, credits: integer)*
 - *Enrolled(sid: integer, cid: integer, grade: string)*
- Physical schema:
 - Relations stored as unordered files.
 - Index on first column of Students.
- External Schema (View):
 - *Course_info(cname: string, enrollment: integer)*

Data Independence *

- Applications insulated from how data is structured and stored.
- Logical data independence: Protection from changes in *logical* structure of data.
- Physical data independence: Protection from changes in *physical* structure of data.

** One of the most important benefits of using a DBMS!*

Concurrency Control

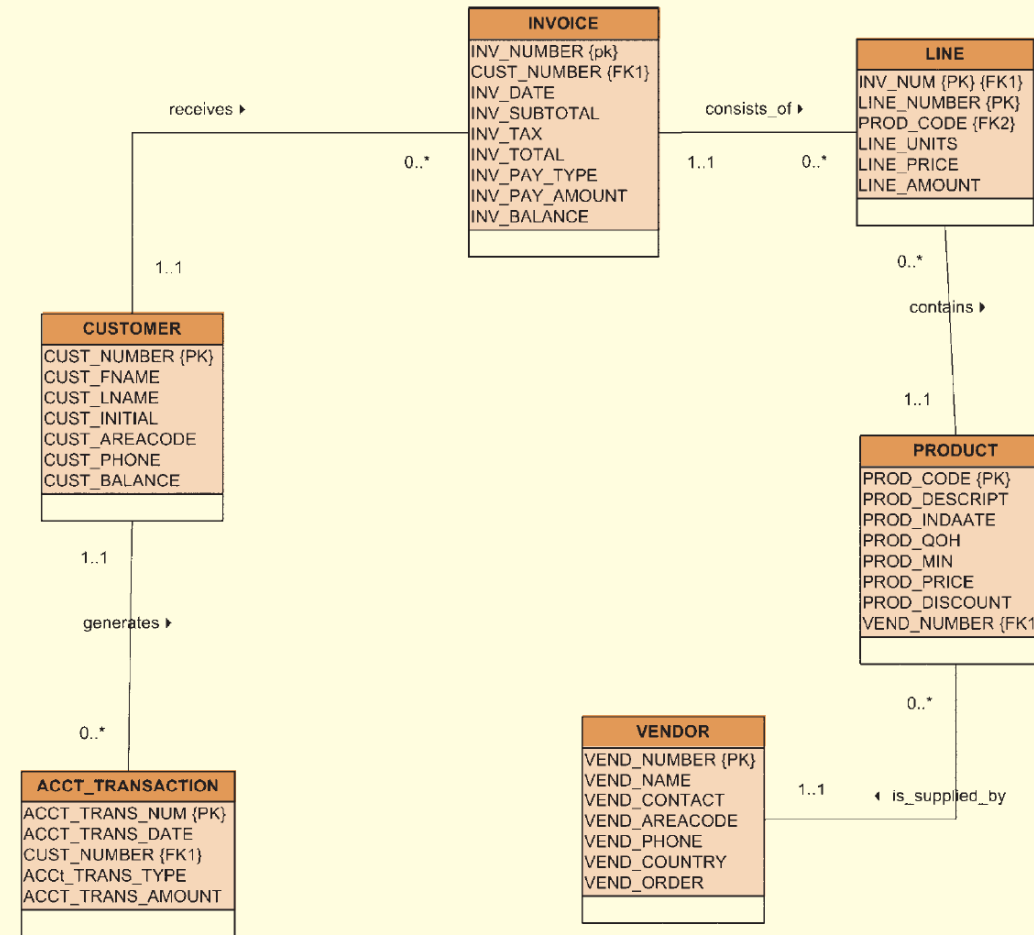
- Concurrent execution of user programs is essential for good DBMS performance.
 - Because disk accesses are frequent, and relatively slow, it is important to keep the cpu humming by working on several user programs concurrently.
- Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed.
- DBMS ensures such problems don't arise: users can pretend they are using a single-user system.

What is a Transaction?

- Logical unit of work that must be entirely completed or aborted
- Consists of:
 - SELECT statement
 - Series of related UPDATE statements
 - Series of INSERT statements
 - Combination of SELECT, UPDATE, and INSERT statements

What is a Transaction?

FIGURE 12.1 The Ch12_SaleCo database ERD



Transaction Properties

Atomicity

- All operations of a transaction must be completed
 - If not, the transaction is aborted

Consistency

- Permanence of database's consistent state

Isolation

- Data used during transaction cannot be used by second transaction until the first is completed

Durability

- Ensures that once transactions are committed, they cannot be undone or lost

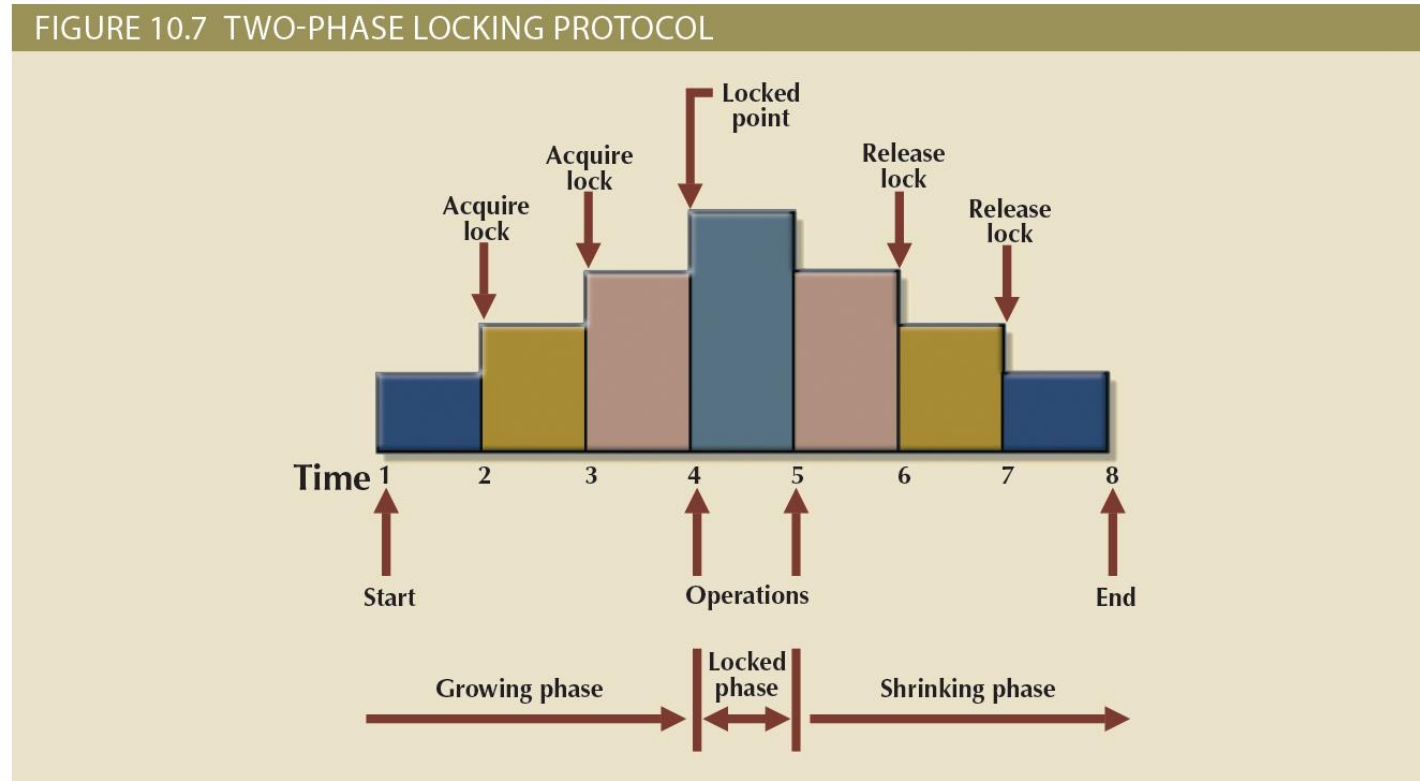
Serializability

- Ensures that the schedule for the concurrent execution of several transactions should yield consistent results

Transaction: An Execution of a DB Program

- Key concept is transaction, which is an *atomic* sequence of database actions (reads/writes).
- Each transaction, executed completely, must leave the DB in a consistent state if DB is consistent when the transaction begins.
 - Users can specify some simple integrity constraints on the data, and the DBMS will enforce these constraints.
 - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
 - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the *user's* responsibility!

Figure 10.7 - Two-Phase Locking Protocol



Scheduling Concurrent Transactions

- DBMS ensures that execution of $\{T_1, \dots, T_n\}$ is equivalent to some serial execution $T_1' \dots T_n'$.
 - Before reading/writing an object, a transaction requests a lock on the object, and waits till the DBMS gives it the lock. All locks are released at the end of the transaction. (Strict 2PL locking protocol.)
 - **Idea:** If an action of T_i (say, writing X) affects T_j (which perhaps reads X), one of them, say T_i , will obtain the lock on X first and T_j is forced to wait until T_i completes; this effectively orders the transactions.
 - What if T_j already has a lock on Y and T_i later requests a lock on Y ? (Deadlock!) T_i or T_j is aborted and restarted!

Ensuring Atomicity

- DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a Xact.
- **Idea:** Keep a log (history) of all actions carried out by the DBMS while executing a set of Xacts:
 - **Before** a change is made to the database, the corresponding log entry is forced to a safe location. (WAL protocol; OS support for this is often inadequate.)
 - After a crash, the effects of partially executed transactions are undone using the log. (Thanks to WAL, if log entry wasn't saved before the crash, corresponding change was not applied to database!)

The Log

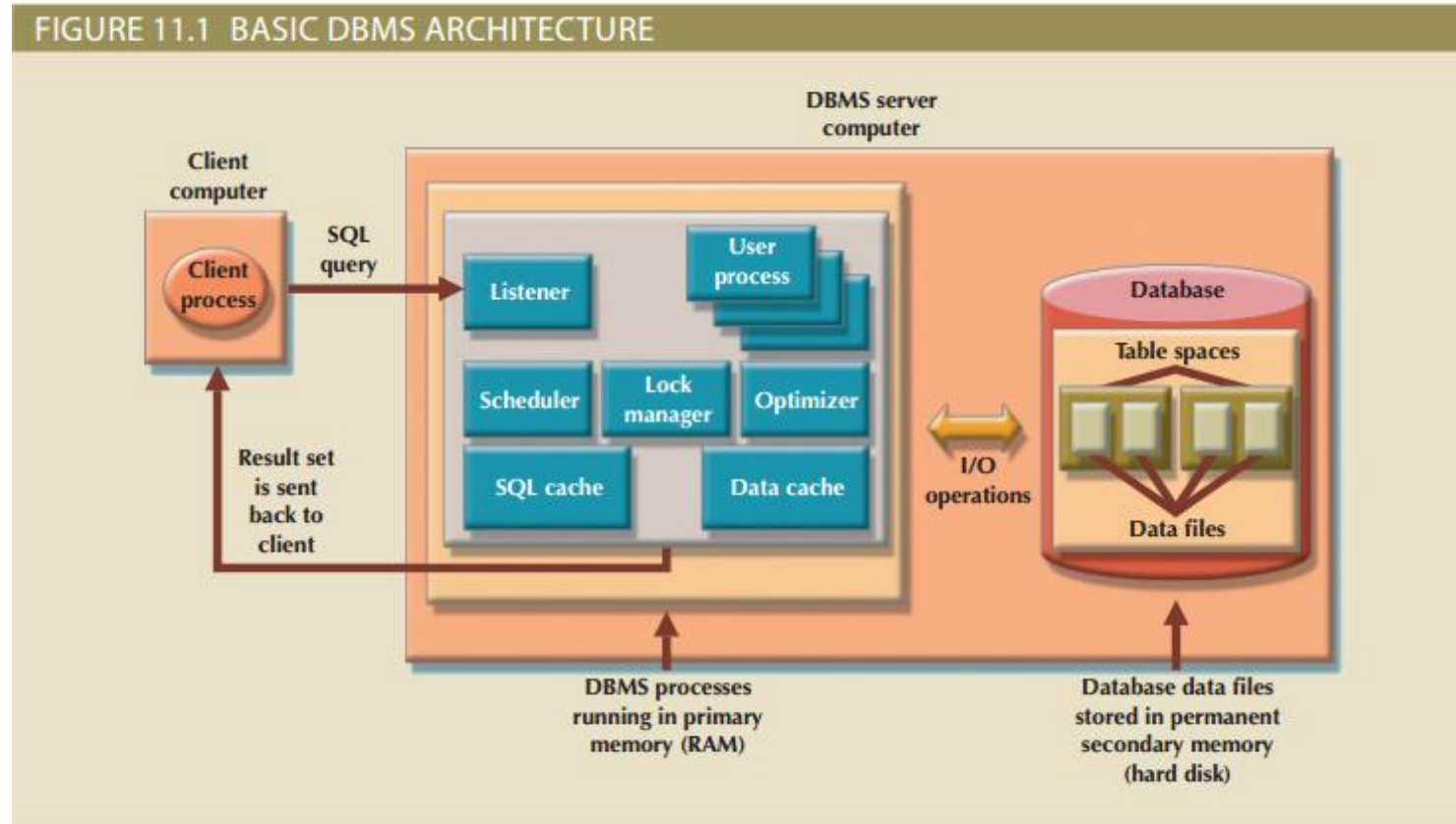
- The following actions are recorded in the log:
 - *Ti writes an object*: The old value and the new value.
 - Log record must go to disk before the changed page!
 - *Ti commits/aborts*: A log record indicating this action.
- Log records chained together by Xact id, so it's easy to undo a specific Xact (e.g., to resolve a deadlock).
- Log is often *duplexed* and *archived* on “stable” storage.
- All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

Databases make these folks happy ...

- End users and DBMS vendors
- DB application programmers
 - E.g., smart webmasters
- Database administrator (DBA)
 - Designs logical /physical schemas
 - Handles security and authorization
 - Data availability, crash recovery
 - Database tuning as needs evolve

Must understand how a DBMS works!

Figure 11.1 - Basic DBMS Architecture



DBMS Architecture

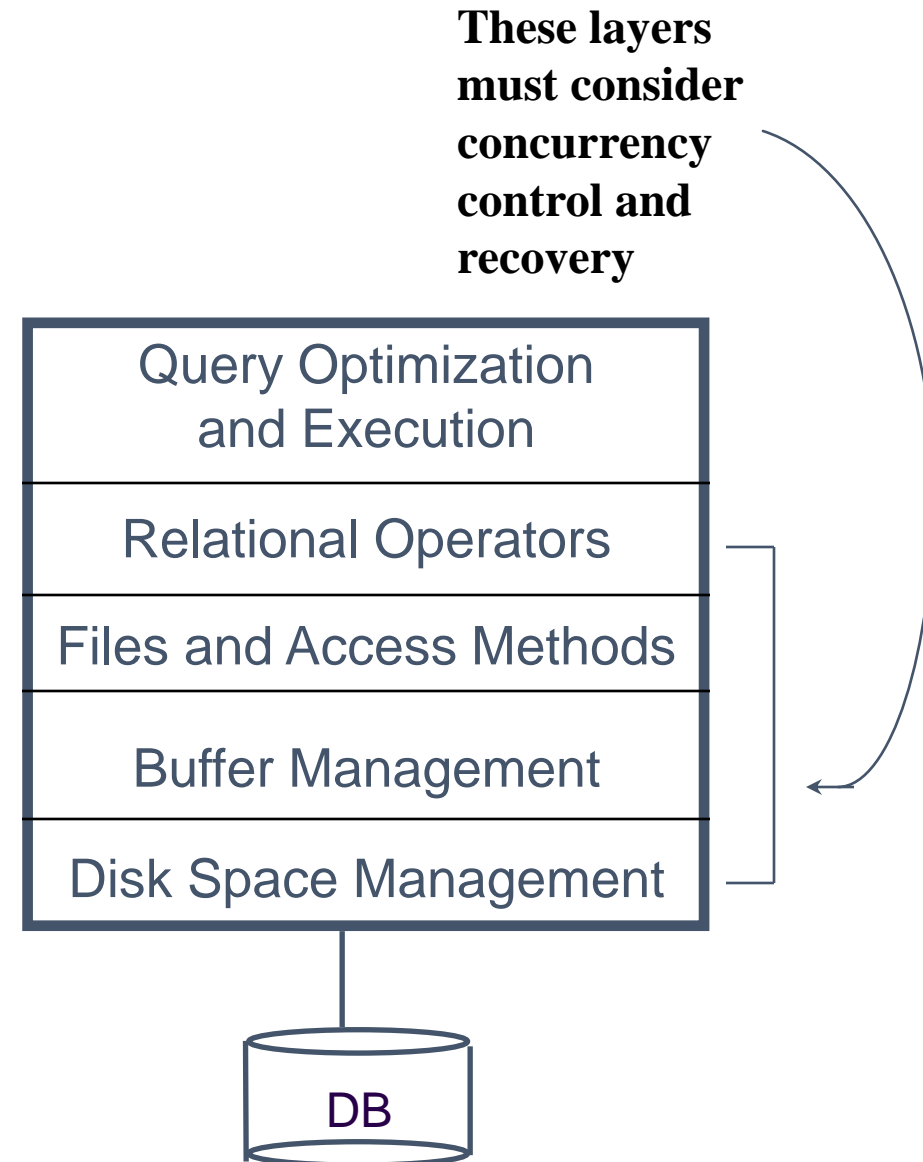
- All data in a database are stored in **data files**
 - Data files automatically expand in predefined increments known as **extends**
- Data files are grouped in file groups or table spaces
 - **Table space** or **file group**: Logical grouping of several data files that store data with similar characteristics
- **Data cache** or **buffer cache**: Shared, reserved memory area
 - Stores most recently accessed data blocks in RAM

DBMS Architecture

- **SQL cache** or **procedure cache**: Stores most recently executed SQL statements or PL/SQL procedures
- DBMS retrieves data from permanent storage and places them in RAM
- **Input/output request**: Low-level data access operation that reads or writes data to and from computer devices
- Data cache is faster than working with data files
- Majority of performance-tuning activities focus on minimizing I/O operations

Structure of a DBMS

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- This is one of several possible architectures; each system has its own variations.



Architecture of a DBMS

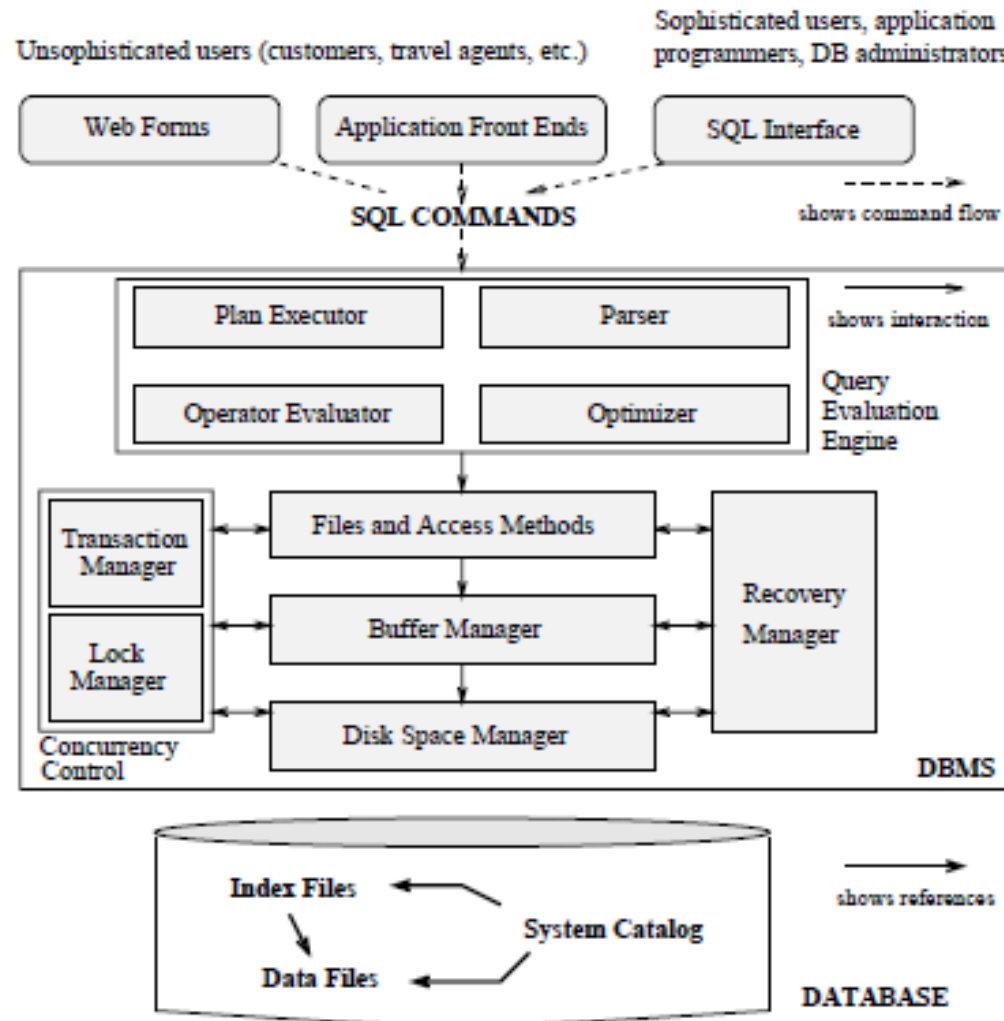


Figure 1.3 Architecture of a DBMS

- The DBMS **accepts SQL commands** generated from a variety of user interfaces, **produces query evaluation plans**, **executes these plans** against the database, and **returns the answers**.
- When a user issues a query, the **parsed query is presented to a query optimizer**, which uses information about how the data is stored **to produce an efficient execution plan** for evaluating the query. An execution plan is a blueprint for evaluating a query, usually **represented as a tree of relational operators** (with annotations that contain additional detailed information about which access methods to use, etc.). Relational operators serve as the building blocks for evaluating queries posed against the data.
- The code that implements relational operators sits on top of **the file and access methods layer**. This layer supports the concept of a file, which, in a DBMS, is a collection of pages or a collection of records. Heap files (files of unordered pages) and indexes are supported. In addition to keeping track of the pages in a file, this layer organizes the information within a page.
- The files and access methods layer code sits on top of **the buffer manager**, which brings pages in from disk to main memory as needed in response to read requests.

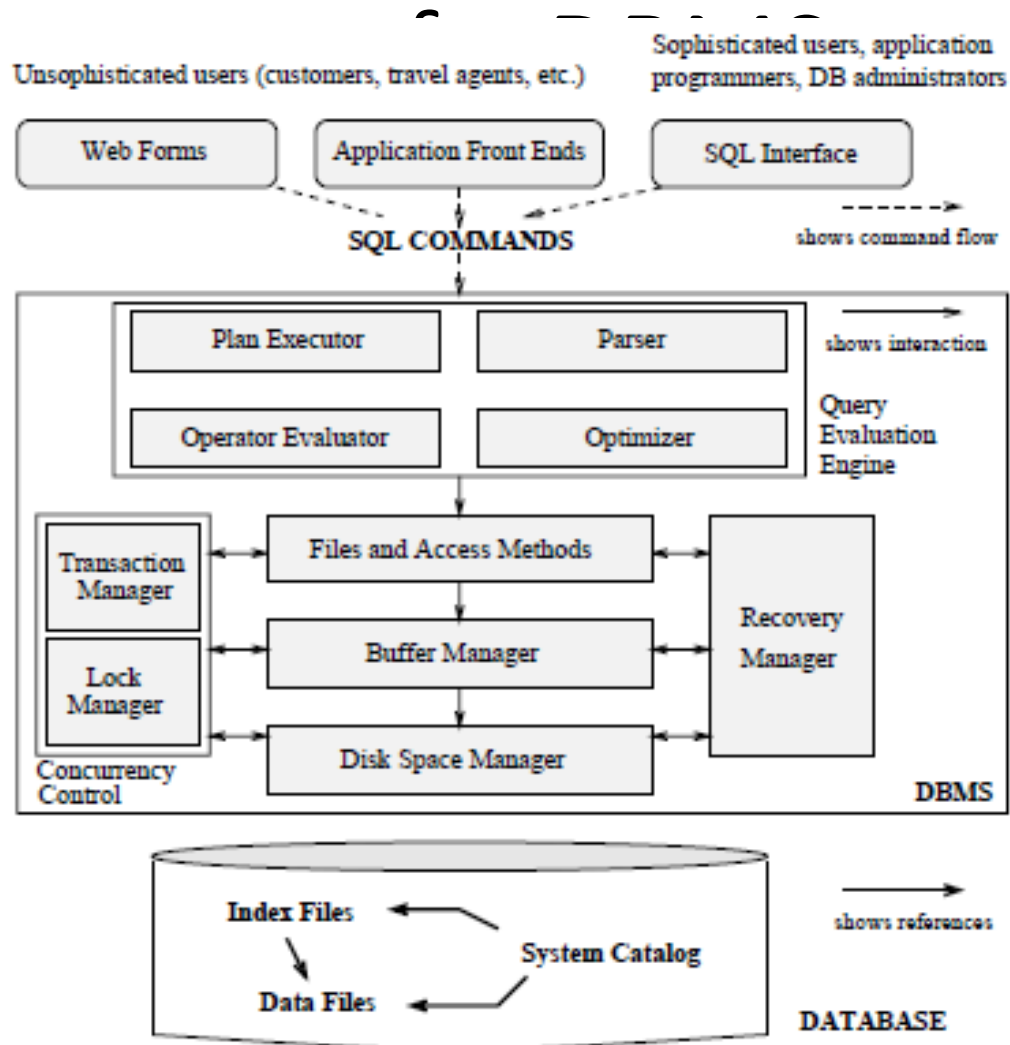


Figure 1.3 Architecture of a DBMS

- The lowest layer of the DBMS software deals with management of space on disk, where the data is stored. Higher layers allocate, deallocate, read, and write pages through (routines provided by) this layer, called the **disk space manager**.
- The DBMS supports **concurrency and crash recovery by** carefully scheduling user requests and maintaining a log of all changes to the database. DBMS components associated with concurrency control and recovery include the
 - transaction manager**, which ensures that transactions request and release locks according to a suitable locking protocol and schedules the execution transactions;
 - lock manager**, which keeps track of requests for locks and grants locks on database objects when they become available; and the
 - recovery manager**, which is responsible for maintaining a log and restoring the system to a consistent state after a crash.
- The disk space manager, buffer manager, and file and access method layers must interact with these components

Summary

- DBMS used to maintain, query large datasets.
- Benefits include recovery from system crashes, concurrent access, quick application development, data integrity and security.
- Levels of abstraction give data independence.
- A DBMS typically has a layered architecture.
- DBAs hold responsible jobs and are **well-paid!** 😊
- DBMS R&D is one of the broadest, most exciting areas in CS.